## SPECIFICATION AMENDMENTS

Please replace paragraph **[0002]** with the following:

**[0002]** As computer server architectures have advanced, more specific functionality ~~have~~ has been added to meet customer needs and to increase uptime. For example, older computer server architectures might employ a single processor that is used to provide substantially all server functionality via execution of firmware and software instructions on the processor, as well as through specific hardware-level logic built into the processor and/or platform. More recently, the single point of service has been discarded for a more distributed service scheme, whereby multiple processors are employed to perform targeted functions.

Please replace paragraph **[0003]** with the following:

**[0003]** For example, modern servers may employ an "out-of-band" management controller that performs separate functions than the servers' primary processor (or processors for multi-processor platforms). Typically, an out-of-band management controller comprises an independent processor, such as a [[base]] baseboard management controller (BMC) or service processor, connected to various hardware components of a server platform to monitor the functionality of those hardware components. For instance, a service processor may be configured to have its own independent link to a network with an independent Internet protocol ("IP") address to allow an administrator on a remote console to monitor the functionality of the server. As used herein, these processors are collectively termed "service processors."

Please replace paragraph **[0006]** with the following:

**[0006]** An administrator working on a remote console 120 coupled to network [[122]] 120 can monitor the functionality of main processor 104, data storage unit 108, or other entities (not shown) via interaction with service processor 102. The functions of service

processor 102 generally include monitoring one or more characteristics or operations of main processor 104 (e.g., monitoring the temperature of processor 104), data storage unit 108, and other hardware components (not shown), recording hardware errors, performing manual tasks initiated by the administrator (such as resetting main processor 104), recovering main processor 104 after an error, performing manual input/output data transfers, and the like. The functions are collectively depicted as services 124.

Please replace paragraph **[0013]** with the following:

**[0013]** Figure 4 is [[a is]] a schematic diagram illustrating the various execution phases that are performed in accordance with the extensible firmware interface (EFI) framework under which the operations of the platform initialization process of Figure 3 may be performed, according to one embodiment of the invention;

Please replace paragraph **[0016]** with the following:

**[0016]** Figure 7 is a schematic diagram of a server having an architecture based on the scalable server management framework of Figure 2 that is enabled to provide data to render a unified presentation of service capabilities on a remote control used by an administrator to request and ~~observer~~ observe server management services.

Please replace paragraph **[0027]** with the following:

**[0027]** In one embodiment, the BUP functionality is facilitated by a firmware component depicted as BUP firmware 222. In the illustrated embodiment, this firmware component is stored as part of the platform's system BIOS 224. As described below, the BUP firmware may be stored elsewhere on the baseboard, or may be loaded from an add-in card or even from a network store.

Please replace paragraph **[0033]** with the following:

**[0033]** Subsequently, a BIOS server management handler is established in a block 320. The handler is used to provide an interface to server management consumers,

as described in further detail below. At the close of the registration process, the pre-boot initialization of the system continues, with the operating system being booted in a block [[320]] 322.

Please replace paragraph **[0034]** with the following:

**[0034]** In accordance with one embodiment, the foregoing service processor registration process may be implemented under an extensible firmware framework known as the Extensible Firmware Interface (EFI) (specifications and examples of which may be ~~found at http://developer.intel.com/technology/efi)~~ published by Intel Corporation). EFI is a public industry specification that describes an abstract programmatic interface between platform firmware and shrink-wrap operation systems or other custom application environments. The EFI framework includes provisions for extending BIOS functionality beyond that provided by the BIOS code stored in a platform's BIOS device (*e.g.*, flash memory). More particularly, EFI enables firmware, in the form of firmware modules and drivers, to be loaded from a variety of different resources, including primary and secondary flash devices, option ROMs, various persistent storage devices (*e.g.*, hard disks, CD ROMs, etc.), and even over computer networks. The current EFI framework specification for the Intel Platform Innovation Framework for EFI specification is entitled, "Intel Platform Innovation for EFI Architecture Specification," version 0.9, September 16, 2003.

Please replace paragraph **[0038]** with the following:

**[0038]** The DXE core is designed to be completely portable with no CPU, chipset, or platform dependencies. This is accomplished by designing in several features. First, the DXE core only depends upon the HOB list for its initial state. This means that the DXE core does not depend on any services from a previous phase, so all the prior phases can be unloaded once the HOB list is passed to the DXE core. Second, the DXE core does not contain any hard coded addresses. This means that the DXE core can be loaded anywhere in physical memory, and it can function correctly no matter where physical memory or where Firmware segments are located in the processor's physical address space. Third, the DXE core does not contain any CPU-specific, chipset specific, or

platform specific information. Instead, the DXE core is abstracted from the system hardware through a set of architectural protocol interfaces. These architectural protocol interfaces are produced by DXE drivers [[104]] 404, which are invoked by DXE Dispatcher [[102]] 402.

Please replace paragraph **[0040]** with the following:

**[0040]** The Boot Services comprise a set of services that are used during the DXE and BDS phases. Among others, these services include Memory Services, Protocol Handler Services, and Driver Support Services: Memory Services provide services to allocate and free memory pages and allocate and free the memory pool on byte boundaries. It also provides a service to retrieve a map of all the current physical memory usage in the platform. Protocol Handler Services ~~provides~~ provide services to add and remove handles from the handle database. [[It]] They also ~~provides~~ provide services to add and remove protocols from the handles in the handle database. Additional services are available that allow any component to lookup handles in the handle database, and open and close protocols in the handle database. Support Services ~~provides~~ provide services to connect and disconnect drivers to devices in the platform. These services are used by the BDS phase to either connect all drivers to all devices, or to connect only the minimum number of drivers to devices required to establish the consoles and boot an operating system (*i.e.*, for supporting a fast boot mechanism). In contrast to Boot Services, Runtime Services are available both during pre-boot and OS runtime operations.

Please replace paragraph **[0047]** with the following:

**[0047]** The DXE drivers that comply with the EFI 1.10 Driver Model do not need to be concerned with this possibility. These drivers simply register the Driver Binding Protocol in the handle database when they are executed. This operation can be performed without the use of any architectural protocols. In connection with registration of the Driver Binding Protocols, a DXE driver may "publish" an API by using the *InstallConfigurationTable* function. [[This]] These published drivers are depicted by API's 418. Under EFI, publication of an API exposes the API for access by other

firmware components. The API's provide interfaces for the Device, Bus, or Service to which the DXE driver corresponds during their respective lifetimes.

Please replace paragraph **[0048]** with the following:

**[0048]** The BDS architectural protocol executes during the BDS phase. The BDS architectural protocol locates and loads various applications that execute in the pre-boot services environment. Such applications might represent a traditional OS boot loader, or extended services that might run instead of, or prior to loading the final OS. Such extended pre-boot services might include setup configuration, extended diagnostics, flash update support, OEM value-adds, or the OS boot code. A Boot Dispatcher [[420]] 422 is used during the BDS phase to enable selection of a Boot target, *e.g.*, an OS to be booted by the system.

Please replace paragraph **[0049]** with the following:

**[0049]** During the TSL phase, a final OS Boot loader [[422]] 424 is run to load the selected OS. Once the OS has been loaded, there is no further need for the Boot Services 406, and for many of the services provided in connection with DXE drivers 404 via API's 418, as well as DXE Services 406A. Accordingly, these reduced sets of API's that may be accessed during OS runtime are depicted as API's 416A, and 418A in Figure 4.

Please replace paragraph **[0054]** with the following:

**[0054]** The boot block contains firmware instructions for performing early initialization, and is executed by processor 202 to initialize the CPU, chipset, and motherboard. (It is noted that during a warm boot early initialization is not performed, or is at least performed in a limited manner). Execution of firmware instructions corresponding to the EFI core are executed next, leading to the DXE phase. As part of initializing the DXE core is initialized, core server management driver 600 is loaded. In turn, this driver is used to initialize the BUP framework, as discussed above with ~~referenced~~ reference to block 302 of Figure 3.

Please replace paragraph **[0055]** with the following:

**[0055]** Henceforth, DXE dispatcher 402 begins loading DXE drivers 404. Each DXE driver corresponds to a system component, and provides an interface for directly accessing that component. Included in the DXE drivers are drivers that will be subsequently employed for registering service processors and supporting OS-runtime server management operations. In Figure 6, these DXE drivers include a DXE driver 602, which is loaded from BMC processor firmware 216, and DXE drivers 604 and 606, which are loaded from add-in service processor firmware 218 and 220, respectively. Loading of DXE drivers 602, 604, and [[604]] 606 causes corresponding API's 608, 610 and 612 to be published by the EFI framework. In one embodiment, data relating to the BUP is stored in a BUP table 508 of the EFI system configuration table (Figure 5).

Please replace paragraph **[0058]** with the following:

**[0058]** In one embodiment, the administrator or similar end-user is enabled to set up use preferences, whereby a service processor having a higher preference among multiple service processors that support like services is selected to perform the service. For instance, Figure [[7]] 8 shows a BUP 226A illustrating one embodiment of a service preference scheme. Under the scheme, an end-user is enabled to set a preferred order of service processors to perform a given task. For example, SERVICE A is supported by each of service processors SM1, SM2, and SM3 (i.e., service processors 204, 206, and 208). It is desired by the end-user to have service processor SM2 perform this task, if available. If service processor SM2 is unavailable, the preference falls to service processor SM3. If neither service processor SM2 or SM3 is available, then service processor SM1 is assigned to perform the service.

Please replace paragraph **[0060]** with the following:

**[0060]** Figure 9 shows a flowchart illustrating operations performed during handling of a service management event, according to one embodiment. The processor begins in response to operations performed in a block 900, wherein a service consumer initiates a server management request. In general, a service consumer may comprise any entity that may request server management services to be performed on its behalf. This includes both humans (e.g., administrators) and programmatic entities (e.g., a software-based server management component). In instances in which the service consumer is an end-user, a software-based service host utility may be employed to provide the end-user with service availability and selection operations, along with corresponding information that is displayed while a service is being performed, such as progress, status, results, data dumps, etc. (not shown).

Please replace paragraph **[0061]** with the following:

**[0061]** In response to the server management request, the BUP framework identifies one or more (as applicable) service processors that are capable of servicing the request, as depicted in a block 902. In block 904, [[If]] if preferences are supported, the BUP framework further filters the selection process based on preferences set up by the end user (such as illustrated in Figure 8). In a block 906, the BUP framework broadcasts the server management request to the relevant service processor(s). In one embodiment in which preferences are not employed, the broadcast is used to access the first available service processor, thus the broadcast is made to all service processors. Under a preference-based scheme, the broadcast (or a unicast) may be targeted toward a selected service processor with the highest preference. The process is completed in a block 908, wherein the service processor(s) service the request and update the BUP of status, results, etc.

Please replace paragraph **[0064]** with the following:

**[0064]** These modifications can be made to the invention in light of the above detailed description. The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification ~~and the~~

~~claims~~. Rather, the scope of the invention is to be determined entirely by the following claims, which are to be construed in accordance with established doctrines of claim interpretation.